Amendments to the Claims

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (currently amended) A method for securing a computer system which comprises ~~comprising~~ at least ~~one~~ a code interpretation module and memory capacities for storing a ~~the~~ interpreted code having measurable physical imprints provided from said code interpretation module, wherein in order to make more difficult attacks ~~said method with the purpose of making attacks~~ based on physical measurements or requiring synchronization with ~~aforesaid~~ said interpreted code, it consists ~~more difficult, consisting~~ of introducing at least two types of alternatives in the execution times of ~~for executing~~ the interpreted code, which have ~~said alternatives having~~ an effect on the execution times of the interpreted code or on its measurable physical imprint, said alternatives being introduced according to at least one of the following steps: a first step of causing at certain places of an interpreted code bypasses towards new portions of code which do not belong to the original code in order to complicate the synchronization and the physical imprint of the execution, and/or a second phase of proposing a plurality of implementations of certain instructions, each requiring a different execution time and/or having a different physical imprint and providing an identical result, so that two executions of one of said certain instruction within a same code may be performed by two different implementations.

2. (currently amended) The method according to claim 1, ~~comprising bypasses towards new code portions, so-called "bypass codes", which do not belong to the original code~~ wherein the said first way in the execution times of the interpreted codes

comprises a first mode of introducing bypass codes including bypass instruction in certain particular locations of the interpreted code, said introducing mode being made at each execution upon generating the interpreted code by a code generator.

3. (cancelled)

4. (currently amended) The method according to claim 1 ~~2~~, wherein said step of causing bypasses comprises ~~comprising~~ a first mode for introducing "bypass codes" consisting of introducing one or more bypass instructions specific to certain particular locations of the code, either manually or automatically during the generation of the aforesaid code.

5. (currently amended) The method according to claim 4,wherein the said bypass instructions are associated with security levels which correspond to complexity levels of their bypass code, the most complex being considered as the most defensive with regard to security attacks requiring synchronization with the code or measurement of its physical imprint.

6. (currently amended) The method according to claim 1 ~~2~~, wherein said step of causing bypasses comprises ~~comprising~~ a second mode for introducing "bypass codes" consisting of introducing the bypass code ~~in~~ during the implementation of the interpreter itself.

7. (previously presented) The method according to claim 6, wherein the bypass code introduced into the implementation of the interpreter is executed either systematically by the interpreter or selectively or randomly.

8. (currently amended) The method according to claim 1 ~~2~~, wherein said step of causing bypasses comprises ~~comprising~~ a first mode for realizing "bypass codes" consisting of performing a so-called "superfluous" calculation depending on data known at execution.

9. (currently amended) The method according to claim 1 ~~2~~, wherein said step of causing bypasses comprises ~~comprising~~ a second mode for realizing "bypass codes" consisting of providing the aforesaid first mode with a random draw of an extra datum during the execution of the superfluous calculation, said extra datum being used in the calculation performed by the bypass code.

10. (previously presented) The method according to claim 8, wherein the aforesaid first mode for realizing "bypass codes" is improved by attaching different security levels to the implementations of instructions and associating them with all the more complex implementations.

11. (currently amended) The method according to claim 1 ~~2~~, wherein said step of causing bypasses comprises ~~comprising~~ a third mode for realizing "bypass codes" consisting of replacing in the aforesaid first and second modes the test for deciding on the next action by a branching in an indirection table containing the addresses of possible actions

at an index calculated from variable items (dynamical datum and/or result from a random draw).

12. (currently amended) The method according to claim 1 ~~2~~, wherein said step of causing bypasses comprises ~~comprising~~ a fourth mode for realizing "bypass codes" consisting of performing a superfluous calculation having the external characteristics of a particular sensitive calculation.

13. (currently amended) The method according to claim 1 ~~3~~, comprising a first mode for introducing a plurality of implementations of certain instructions consisting of enriching the set of instructions recognized by the interpreter with a plurality of implementations for a given instruction; the aforesaid instructions are performed either manually by programming or automatically upon code generation.

14. (currently amended) The method according to claim 1 ~~3~~, comprising a second mode for introducing the aforesaid plurality of implementations of certain instructions consisting of comprising in the actual implementation of the instruction, a branching to a portion of at least one alternative code with a variable physical imprint or duration, which dynamically determines the implementation to be executed.

15. (previously presented) The method according to claim 14, comprising a first mode for realizing the aforesaid alternative code consisting of proposing a plurality of different implementations of the instruction and by conditioning the choice of the executed version to a dynamical test, i.e., depending on data known at execution.

16. (previously presented) The method according to claim 14, comprising a second mode for realizing the aforesaid alternative code consisting of improving the aforesaid first mode for realizing "alternative codes" by providing it with a random draw for achieving the test leading to the dynamical choice of the executed version.

17. (previously presented) The method according to claim 14, comprising a third mode for realizing the aforesaid "alternative code" consisting of improving the aforesaid first and second modes for realizing "alternative codes" consisting of replacing the test for deciding on the selected version with a branching in an indirection table containing the addresses of the available version at an index calculated for variable items.

18. (previously presented) The method according to claim 1, being implemented on a module for interpreting software code, a so-called virtual machine.

19. (previously presented) The method according to claim 18, wherein said virtual machine is a Java platform.

20. (previously presented) The method according to claim 1, being implemented on a module for interpreting physical code.

21. (previously presented) The method according to claim 1, being implemented on an embedded system and on an interpretation module of the microcontroller or microprocessor type.